Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent is an iterative method for optimizing an objective function with suitable smoothness properties (differentiable or subdifferentiable). It replaces the actual gradient by an estimate thereof. Especially in high-dimensional optimization problems this reduces the very high computational burden, achieving faster iterations in trade for a lower convergence rate

we have empirical evidence that both computing the gradient for just one sample at a time and doing it for all the samples in the same call is worse than using minibatch

Require: Learning rate $\eta \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  while stopping criterion not met **do**
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset D → m random samples from the dataset at each iteration
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} - \eta \mathbf{g}$
    $k \leftarrow k + 1$
  end while

derivative of the sum is the sum of the derivatives

mean of the loss

learning rate

Observe: $\nabla_{\boldsymbol{\theta}} L(f(\mathbf{x}; \boldsymbol{\theta}), \mathbf{t})$ obtained with backprop

At each step of the algorithm we run m instances of backpropagation. We repeat this until the stopping criterion is met. What we do is: we run m instances of backpropagation to compute the gradient for each sample in the minibatch and then do the average of all these numbers

while ! stopping-criterion : $\sim 10^4$
  backpropagation $\sim 10^2$
} you run backpropagation in the order of 10^6

When you are far from the solution η should be large enough so that you speed up the computation; when you are close to the solution η should be small.
On the other hand, if η is too large you can diverge, if η is too small the computation will take too much time to converge.
η usually changes according to some rule through the iterations

until iteration τ $(k \leq \tau)$ $\quad \eta^{(k)} = \left(1 - \frac{k}{\tau}\right) \eta^{(k)} + \frac{k}{\tau} \eta^{(\tau)}$

after iteration τ $(k > \tau)$ $\quad \eta^{(k)} = \eta^{(\tau)}$

value decreasing for the first τ iterations. After iteration τ you keep the value $\eta^{(\tau)}$ constant

How many iterations τ? When will the learning rate be fixed?
This is another thing to consider; bad choices will lead to similar problems as before

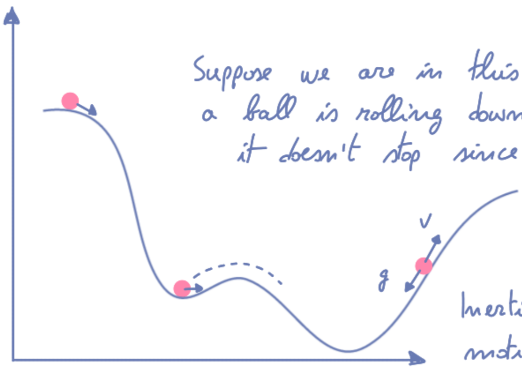
SGD with momentum

Momentum can accelerate learning
Motivation: Stochastic gradient can largely vary through the iterations

Require: Learning rate $\eta \geq 0$
Require: Momentum $\mu \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
  $k \leftarrow 1$
  $\mathbf{v}^{(1)} \leftarrow 0$
  while stopping criterion not met **do**
    Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset D
    Compute gradient estimate: $\mathbf{g} = \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
    Compute velocity: $\mathbf{v}^{(k+1)} \leftarrow \mu \mathbf{v}^{(k)} - \eta \mathbf{g}$, with $\mu \in [0, 1)$
    Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \mathbf{v}^{(k+1)}$
    $k \leftarrow k + 1$

velocity computed at the previous step

$\theta^{(k)} + \mu v^{(k)} - \eta g$ while before we had $\theta^{(k)} - \eta g$
even if $g = 0$ (thus $\eta g = 0$) we still move of $\mu v^{(k)}$
in the same direction of the solution



Suppose we are in this situation:
a ball is rolling down a hill; when the ball reaches a local minima
it doesn't stop since it still has some inertia that can be enough
(depending on the friction and other physical parameters)
to overcome it; the ball can procede to the global minima

Inertia: the idea that an object will continue its current
motion until some force causes its speed or direction to change

The idea of momentum in SGD is considering the direction in which the solution is moving
and to keep moving it in that direction even after a local minima for a little.

Classic SGD will immediately stop while SGD with momentum will not.

In SGD with momentum we have two parameters $\begin{cases} \eta & \text{learning rate} \\ \mu & \text{momentum} \end{cases}$

We will reach a point in which v and g are equal.
In the following iteration g will overcome v and
the direction will be inverted

High μ can introduce high oscillations, we need to be careful.
This is why also momentum μ might change according to some rule
through the iterations

SGD with Nesterov momentum

Momentum can accelerate learning
Motivation: Stochastic gradient can largely vary through the iterations

Require: Learning rate $\eta \geq 0$
Require: Momentum $\mu \geq 0$
Require: Initial values of $\boldsymbol{\theta}^{(1)}$
$k \leftarrow 1$
$\mathbf{v}^{(1)} \leftarrow 0$
while stopping criterion not met **do**
Sample a subset (minibatch) $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(m)}\}$ of m examples from the dataset D
3 Compute gradient estimate: $\mathbf{g} = \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \boldsymbol{\theta}^{(k)}), \mathbf{t}^{(i)})$
1 Compute velocity: $\mathbf{v}^{(k+1)} \leftarrow \mu \mathbf{v}^{(k)} - \eta \mathbf{g}$, with $\mu \in [0, 1)$
2 Apply update: $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + \mathbf{v}^{(k+1)}$
$k \leftarrow k + 1$

Momentum is applied before computing the gradient

$$\tilde{\theta} = \theta^{(k)} + \mu v^{(k)}$$

$$g = \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), t^{(i)})$$

Sometimes it improves convergence rate

At this moment we have no theoretical proof that one approach is always better than the other $\longrightarrow$ empirical evaluation for the particular problem

Algorithms with adaptive learning rates

Based on the analysis of the gradient of the loss function, several methods can automatically compute the SGD paramethers (determine if the learning rate should be increased or decreased). These methods are called optimizers.

Some examples:

- AdaGrad
- Adam
- RMSProp
- ...

A stands for "adaptive"